Средства анализа параллельных SPMD программ

Яковенко П.Н.

Аннотация. В этот статье представлен обзор задач, методов и средств анализа производительности и масштабируемости SPMD программ для параллельных вычислительных систем с распределенной памятью. Рассмотрены такие вопросы отладки параллельных программ как визуализация различных аспектов выполнения программы, моделирование параллельной программы при помощи алгебраических формул, зависящих от размерности задачи и числа процессоров, мониторинг (контролируемое выполнение). Обзор основан на большом количестве отладочных средств, разрабатываемых в рамках академических проектов, а также коммерческих продуктов. Ссылки на все эти средства анализа параллельных программ присутствуют в библиографии

1. Введение

Достижения в области микропроцессорных и сетевых технологий позволили конструировать параллельные вычислительные системы с большим числом процессоров (например, Cray T3E, IBM SP2, Intel Paragon, кластеры на базе сети Marynet и SCI). К сожалению приложения, разработанные для традиционных последовательных компьютеров или даже для векторных суперкомпьютеров, не могут быть автоматически перенесены на подобные системы в силу архитектурных различий.

Производительность параллельной программы зависит от большого числа параметров как непосредственно самой задачи, так и компьютерной системы, библиотеки поддержки выполнения и коммуникационной среды. Аккуратное предсказание влияния тех или иных стратегий на производительность программы, а также то, как скорость выполнения программы будет меняться при различных параметрах задачи и вычислительной системы, коренным образом может сократить время разработки эффективного программного обеспечения.

Помимо описанных выше трудностей разработчик сталкивается с проблемой, заложенной в самой природе параллельной программы, - недетерминированностью выполнения [4]. При каждом выполнении программы сообщения от взаимодействующих процессов могут приходить в

разном порядке. Как следствие, программа на одних и тех же входных данных может давать разные результаты. Присутствие в программе межпроцессного взаимодействия неминуемо повлечет за собой возникновение коммуникационных ошибок (неправильное указание отправителя или получателя), блокировок процессов (ошибки синхронизации), длительных простоев процессов в ожидании событий в системе, которые потребуется устранять на стадии отладки программы.

Все описанные выше проблемы требуют наличия у разработчика мощного набора инструментов, позволяющих с той или иной степенью автоматизации эффективно отлаживать параллельные программы. К таким средствам относятся стандартные отладчики, утилиты, позволяющие выполнять программу детерминировано, визуализировать ее выполнение и обмен сообщениями между процессами, оценивать ее масштабируемость при помощи соответствующих алгебраических моделей, устранять лишние временные задержки, возникающие в ходе выполнения программы. Более того, недетерминированность выполнения параллельной программы требует от разработчика глубокой внимательности при использовании каких-либо отладочных средств, т.к. любое вмешательство в выполнение исходной программы, влияющее на ее временные характеристики, может привести к тому, что программа в ходе отладки будет функционировать совсем не так, как до и после нее [5].

2. Масштабируемость параллельной вычислительной системы

В этом разделе рассматриваются вопросы масштабируемости параллельных программ. Вводятся определения необходимых терминов, формулируются законы Амдала и Густафсона-Барсиса, устанавливаются необходимые условия создания высокопроизводительной масштабируемой параллельной программы.

2.1 Терминология

Современные вычислительные системы с распределенной памятью состоят из вычислительных узлов, каждый из которых может содержать следующие устройства:

- один или несколько центральных процессоров (обычно RISC);
- локальную память (прямой доступ к памяти других узлов невозможен);
- коммуникационный процессор или сетевой адаптер;
- жесткие диски и/или другие устройства ввода/вывода.

Узлы связаны между собой через некоторую коммуникационную среду (высокоскоростная сеть, коммутатор и т.п.).

При проектировании параллельной программы основная задача состоит создании высокопроизводительной масштабируемой программы. В соответствии с толковым словарем русского языка производительность - есть эффективность трудовой деятельности, т.е. высокопроизводительная программа должна быть эффективной.

Определение 1. Параллельная программа называется эффективной на данной параллельной вычислительной системе, если накладные расходы ограничены и достаточно малы.

Определение 2. Параллельная вычислительная система называется масштабируемой, если производительность системы пропорциональна сумме производительностей всех узлов.

Для каждой масштабируемой вычислительной системы существует максимальное число узлов (оно определяется архитектурой системы) такое, что при большем числе узлов система перестает быть масштабируемой.

Определение 3. Параллельная программа называется масштабируемой, если скорость ее выполнения пропорциональна производительности вычислительной системы.

Для масштабируемых вычислительных систем и параллельных программ коэффициент пропорциональности должен быть близок к единице. На практике, в силу накладных расходов, с увеличением числа процессоров, рост производительности замедляется.

Для создания высокопроизводительной и масштабируемой параллельной программы необходимо выполнить ряд условий:

- разработать эффективный алгоритм, обладающий высокой степенью параллелизма;
- обеспечить равномерную вычислительную нагрузку всех узлов;
- минимизировать накладные расходы на управление параллелизмом (синхронизация процессов и т.п.);
- минимизировать накладные расходы, вызванные необходимостью коммуникационных обменов между процессами (пересылка сообщений, групповые операции и т.д.).

Масштабируемость является одной из наиболее важных характеристик программной системы, зачастую играющей большую роль, чем ее производительность. Более того, понятия "производительность" и "масштабируемость" тесно связаны между собой, и по масштабируемости системы можно судить о ее производительности [1].

2.2 Законы Амдала и Густафсона-Барсиса

Производительность и масштабируемость параллельной программы легко описать при помощи кривых ускорения (speedup curves). Точка кривой ускорения вычисляется посредством деления времени, необходимого для решения данной задачи с использованием одного процессора, на длительность вычисления решения при использовании N процессоров. Закон Амдала, сформулированный в 1967 г., устанавливает зависимость между объемом последовательного ($b,0<=b<=1$) и параллельного кода в программе и ускорением S, достигаемым при использовании N процессоров вместо одного для решения данной задачи [2]:

$$S = \frac{N}{N * b + 1 - b}$$

Полученная формула является серьезным сдерживающим фактором для программистов, склоняющихся к созданию параллельных программ, поскольку она обосновывает использование параллельных алгоритмов только для тех задач, у которых b является ничтожно малой величиной (например, задач линейной алгебры).

Закон Амдала формулируется для общего случая параллельной программы, однако если рассматривать более узкий класс программ, то можно получить более оптимистичные оценки. Закон Густафсона-Барсиса исходит из того, что параметры b и N не являются независимыми [3]:

$$S = N - (N - 1) * b$$

Действительно, запуск одной и той же программы на разном числе процессоров разумно производить только в академических целях. На практике, размерность задачи масштабируется при изменении количества доступных процессоров. Пользователи обычно имеют контроль над такими параметрами, как шаг сетки, длина временного интервала, величина погрешности и пр., и подбирают эти параметры, чтобы время выполнения программы было допустимым. Более верным будет считать, что время выполнения, а не размерность задачи является постоянным.

При N = 32 и b=0.2 будет достигаться максимальное ускорение в 25.8 раз, что почти в 6 раз быстрее, чем ускорение, достигаемое в законе Амдала. Следует, конечно, помнить, что эта формула не учитывает коммуникационных задержек, накладных расходов операционной системы, таких как создание процесса, управление ресурсами памяти, буферизацию сообщений и т.д.

Очевидно, что SIMD и SPMD формы параллелизма естественным образом укладываются в модель, предложенную Густафсоном и Барсисом.

Весь спектр задач, возникающих при отладке, анализе и доводке распределенной масштабируемой SPMD программы, можно условно поделить следующим образом:

- визуализация различных аспектов выполнения программы;

- моделирование программы при помощи алгебраических формул, зависящих от размерности задачи и числа процессоров, для анализа производительности и масштабируемости;
- отладка (мониторинг) параллельной программы.

В ходе решения этих задач возникают дополнительные подзадачи, связанные с обеспечением детерминированного выполнения программы, упорядочиванием событий произошедших в разных процессах, хранением собранной трассировочной информации и т.д.

3. Визуализация

В этом разделе рассматриваются неотъемлемые свойства "полезной" системы визуализации, подзадачи, возникающие в ходе построения графического представления тех или иных аспектов выполнения параллельной программы, а также различные методы визуализации поведения программы и ее статистических характеристик.

Текстовые представления данных, описывающие выполнение параллельной компьютерной системы, по своей природы последовательны и достаточно тяжелы для усваивания информации. В то же время, графические визуализации становятся все более мощными и удобными инструментами для понимания поведения системы и выполнения программы, а также для целей отладки и характеризации производительности системы.

3.1 Информативность визуализации

Визуализация - двойственный метод, при правильном использовании он позволяет быстро ухватывать главные моменты, в противном случае может принести ужасающие результаты. К сожалению, нарисовать привлекательную картинку намного проще, чем полезную. Графическое изображение является более емким, чем текстовое представление данных. Информация может представляться посредством различных графических примитивов (линий, стрелок, окружностей и т.д.), трехмерных объектов, иконок и пр. Элементы одного типа различаются цветом, толщиной линий, формой линий (сплошная, прерывистая, двойная). Все эти средства дают возможность разместить на экране монитора огромный объем информации, позволяющий одним взглядом оценить различные характеристики параллельной программы. Неправильное их использование сделает систему визуализации совершенно неинформативной для пользователя или вообще сформирует у него некорректное представление об отлаживаемой программе.

Статья [36] суммирует современные подходы по графическому отображению информации и предлагает технологический и теоретический базис для будущих исследований в области средств визуализации производительности параллельных программ. В статье [16] предлагается набор критериев, которым должна удовлетворять система визуализации. Наиболее важным следует

считать то, что такая система, прежде всего, должна направлять, информировать, а не иллюстрировать известные факты. Разница между информированием и иллюстрированием состоит в различии представления данных. Если система отображает полезные, информативные картинки без какого-либо серьезного вмешательства со стороны пользователя, то у нее есть хороший потенциал. Если же пользователю требуется на каждом шагу направлять систему для выборки и обработки данных, то можно сказать, что он имеет глубокое представление о структуре программы. Тем самым роль такой системы сводится к графическому представлению уже известной информации. Соответственно и полезность такой системы мала.

3.2 Задачи визуализации

Исследования в области визуализации программ можно рассматривать на двух уровнях: с точки зрения выполняемой задачи и цели, поставленной разработчиками данного средства графического представления. При создании двух- или трехмерного образа системы решается целый ряд неотъемлемых задач, таких как сбор информации, анализ информации, сохранение исходных и обработанных данных и непосредственно графическое изображение.

Целью графического представления может быть отладка программы или системы, вычисление и оптимизация производительности или своего рода "электронное документирование" посредством отображения структур данных, анимации алгоритма или других графических изображений, которые выражают знания о функционировании системы, программы или алгоритма. Цели визуализации программы могут налагать дополнительные ограничения на выполняемые задачи. Поэтому "цели" можно рассматривать как второй уровень характеризации визуализирующей системы.

3.3 Сбор данных

Множество параметров и большой объем информации, обрабатываемый при параллельных вычислениях, требует, чтобы инструментирующий код для сбора данных использовался крайне избирательно и аккуратно. Инструментирование может происходить на аппаратном и на программном уровне. Код может вставляться для решения задач визуализации и затем удаляться из окончательной версии, существовать постоянно внутри системы и активироваться по требованию или быть динамически добавляемым, используя специализированные приемы отладчиков.

Любое инструментирование в большей или меньшей степени влияет на производительность изучаемой системы. Степень такого влияния, или так называемый эффект зондирования (probe effect), определяется объемом собранной информации, оборудованием мониторинга, системным программным обеспечением, архитектурой системы и парадигмой программирования [18]. Этот эффект выражается в том, что дополнительные накладные расходы на выполнение инструментальной части программы могут

нарушить исходную последовательность событий в отлаживаемой программе. Как следствие, временные характеристики системы искажаются и ошибка, для поиска которой проводилось инструментирование, может не проявиться во время отладки.

Эти факторы вкупе с целью визуализации определяют тип используемого инструментирования.

3.4 Инструментирование

Информация, которая может быть получена при помощи аппаратуры, включает такие показатели как количество инструкций в секунду, количество операций с плавающей точкой в секунду, количество попаданий в кэш, пропускная способность памяти и коммутаторов, а также адреса инструкций и данных в памяти. В качестве примера можно привести микропроцессор мониторинга для параллельного процессора RP3 компании IBM [19]. Для визуализации алгоритма требуются более высокоуровневые данные. Они могут быть получены при помощи программного инструментирования.

Программное инструментирование состоит в помещении небольших кусков кода внутрь операционной системы, системы поддержки выполнения или прикладной программы. Функция этого кода, называемого сенсором заключается в посылке некоторого значения компоненту, ответственному за суммирование всей информации. Уровень, на котором размещается сенсор, определяет тип собираемой информации. Инструментирование операционной системы может использоваться для фиксации данных об отправляемых и получаемых сообщениях, создании процессов, окончании программы, подкачке виртуальной памяти, переключении контекстов, доступе к памяти и системных вызовах.

Инструментирование системы поддержки выполнения обеспечивает информацией о состоянии различных очередей, приобретение и освобождение захватов, вход и выход из критических секций, прибытие и проход через барьеры, вход и возврат из процедур. Подобная информация может быть использована для выявления простоев в программе применительно к конкретным участкам кода.

Инструментирование прикладной программы дает доступ к абстрактным, высокоуровневым событиям и конструкциям внутри прикладной программы. Инструментирование на прикладном уровне обеспечивает достаточно детальную картину поведения программы для целей отладки, позволяя пользователю проверять как корректность всей программы, так и отдельных ее участков.

Добавление сенсоров в программу может до определенной степени быть автоматизировано [37, 20]. Библиотека инструментирования Pablo [21] предоставляет возможность пользователю определять свои функции, причем информация о событиях в системе, прежде чем попасть в трассировочный файл, будет передана этим функциям. Данные в трассировочный файл

попадают в формате SDDF, содержащем описание записей непосредственно в самом файле. Это позволяет легко добавлять в файл записи нового типа. Система Paradyn [22] инструментирует программы динамически. Пользователь указывает интересующие его события уже во время выполнения программы. Разработчики нацеливают Paradyn на доводку очень больших систем, которые выполняются несколько дней и более. В таких ситуациях повторный запуск приложения является неприемлемым с точки зрения времени, необходимого для получения трассировочной информации.

Другой подход состоит в инструментировании библиотек, реализующих функции параллельного программирования. Коммуникационная библиотека PICL [17] генерирует файл событий, таких как отправка и получение сообщения, а также статистика вычислений и коммуникационных операций. Трассировочный файл формата PICL используется многими системами визуализации производительности параллельных программ, например Paragraph [23].

3.5 Детерминированное выполнение

Некорректная параллельная программа, вследствие недетерминированного выполнения, на одних и тех же входных данных может давать разные результаты. Для локализации ошибки требуется обеспечить идентичность выполнения последовательных запусков программы на одном наборе данных.

Проблема воссоздания исходного поведения параллельной программы на данном наборе исходных данных при каждом последующем запуске решается с использованием механизма "записи и воспроизведения", который заключается в трассировке всех событий, влияющих на детерминированность выполнения. В распределенной параллельной программе недетерминированность порождается механизмом обмена сообщениями между процессами. Во время первого (эталонного) выполнения программы - стадии записи - отладочная система записывает в трассировочный файл всю необходимую информацию о передаваемых сообщениях. Данный файл может быть использован при последующих выполнениях программы - стадиях воспроизведения - для воссоздания поведения эквивалентного эталонному.

Система CPDE [39] при компиляции генерирует две программы: одна содержит специальный код для записи всех событий в системе в трассировочный файл, другая версия - код для выполнения программы с сохранением последовательности событий. Статья [38] предлагает для программ на базе библиотек MPI и PVM сохранять минимум информации во внешний файл, тем самым, уменьшая его размер и влияние эффекта зондирования. В статье обосновывается механизм, позволяющий записывать только номер процесса отправителя в каждом событии получения сообщения, причем рассматриваются только те события, где сообщение принимается по маске, т.е. отправителем может быть любой процесс программы, или любой процесс в группе.

3.6 Анализ данных

Анализ данных является неотъемлемой фазой при визуализации параллельной программы. Эта фаза включает в себя такие действия, как вычисление различных статистик, упорядочение событий, поступивших от разных процессоров, обнаружение в потоке более высокоуровневых событий.

Наиболее общим для всех целей визуализации является упорядочивание событий в потоке. Поскольку время на разных процессорах, вообще говоря, не совпадает, то поток событий должен быть упорядочен некоторым образом. В противном случае может возникнуть ситуация, при которой сообщение было отправлено раньше, чем его получили.

В распределенной системе в силу отсутствия глобального времени события упорядочиваются частично. Предложенный Лампортом механизм причинной упорядоченности [24] на основе отношения "произошло до" позволяет избежать казусов, когда сообщение принимается до его отправки. При таком механизме события, произошедшие на одном процессоре, упорядочиваются посредством физического времени данного процессора, т.е. в пределах одного процессора достигается полный порядок. События, произошедшие на разных процессорах, упорядочиваются через механизм сообщений. Событие отправки сообщения происходит непосредственно перед событием получения сообщения. Распространяя это правило транзитивно на все процессоры можно получить частичный порядок всех событий в системе. Такое упорядочивание также называют логическим временем.

Существуют также другие механизмы упорядочивания событий. К ним можно отнести фазовое время [25], при котором события упорядочиваются только внутри своей фазы. Например, все события передачи сообщения 32 процессору объединяются в одну фазу, внутри которой они упорядочены. Статья [40] предлагает использовать виртуальное время. Виртуальное время организовано таким образом, что во время работы программы оно совпадает с физическим, а во время работы сенсора мониторинга - останавливается. Такой подход позволяет сохранить интервалы времени между событиями в инструментированной системе неизменными по отношению к ее исходному варианту.

Следует сказать, что подход в упорядочивании событий сильно зависит от целей системы визуализации. В большинстве случаев достаточно лампортовского частичного порядка, в то время как в некоторых ситуациях, например, таких как анализ производительности параллельной программы, необходимо наличие физического времени. Разумеется, в распределенной системе организовать глобальное время практически невозможно, однако использование программных механизмов коррекции времени позволяет с высокой точностью приблизить логическое время к физическому.

3.7 Хранение данных

Требования к хранению собранных в ходе мониторинга данных определяются большей частью предполагаемым будущим использованием сохраненной информации.

Системы, предназначенные для визуализации исключительно поведения системы, например анимированного проигрывания событий в программе, предъявляют крайне слабые требования к входным данным. Чаще всего это обычный двоичный файл, возможно нерегулярной структуры, который грузится в память и по нему скользит окно, определяющее ту информацию, которая отображается на экране.

Некоторые системы позволяют проводить достаточно сложные операции с трассировочными данными, такие как поиск, сортировка и другие формы анализа. Статья [41] представляет проект Prophesy - инфраструктуру из трех реляционных баз данных, позволяющую хранить информацию о выполнении различных версий программы, строить аналитические модели, изучать масштабируемость, влияние функций ввода/вывода на производительность системы и т.д.

Среда SIEVE [26] - система визуализации масштабируемых параллельных программ - основана на электронных таблицах. Пользователь может вывести информацию в виде таблицы, где столбцы содержат показатели для строк - процессоров вычислительной системы.

3.8 Графическое представление

Графическое представление информации, особенно статистической, является областью исследований, ведущей свои истоки к началу двадцатого столетия. По аналогии с другими задачами, обсуждаемыми в этой работе, аспекты графического представления определяются нацеленностью конкретной системы визуализации. Отладчики отображают некоторым образом состояние программы. Часто, это узко специализированное представление некоторого аспекта вычислений, такого, как шаблон доступа к общей памяти, межпроцессные взаимодействия через некоторый канал или порядок вызова подпрограмм. Средства визуализации производительности системы обычно отображают стандартные метрики, среди которых загрузка процессора, использование виртуальной памяти, коммуникационная нагрузка и т.д. Системы визуализации поведения программы обеспечивают графическое представление структур данных на высоком уровне, операций, обновляющих эти структуры данных, а также некоторое абстрактное представление вычислений происходящих в программе и процент ее завершенности. Такие визуальные представления зачастую не только удобны в плане понимания вычислительной структуры программы, но и полезны с точки зрения отладки и вычисления производительности системы.

Вычислительная структура отображается, как правило, в виде графов, в которых узлы представляют сущности программы, а дуги зависимости между

ними или временной порядок. Часто, такие отображения могут быть анимированными, выделяя текущую сущность. Такие визуализации могут быть полезными при анализе поведения программы, позволяя пользователю выявлять некорректные последовательности вызовов подпрограмм или невызываемые подпрограммы. Анимированный граф также раскрывает время присутствия в конкретной процедуре. Например, дисплей задач в Paragraph [23] может быть использован именно для решения таких задач.

Среди других механизмов следует отметить карту параллельности [27] и граф причинности Макбилана [28]. HeNCE система [29] представляет собой интегрированную визуальную среду для создания, компиляции, выполнения и анализа PVM программ. Во время выполнения программы HeNCE может отображать ее анимированное поведение в виде графа. Узлы графа вычислений изменяют цвет для индикации возникновения различных трассировочных событий.

Для графического представления коммуникационных взаимодействий между процессорами чаще всего используются различные графические примитивы, такие как прямоугольники, стрелки и линии. Прямоугольники обычно изображают процессоры, линии - соединения между ними, а стрелки - коммуникационные события. В некоторых случаях расположение прямоугольников относительно друг друга раскрывает топологию процессоров в вычислительной системе. Такая информация может быть статически заложена в коммуникационной библиотеке, если вычислительная система имеет фиксированную топологию.

Некоторые системы поддержки выполнения позволяют задавать топологию в программе логическим образом. Например, библиотека MPI содержит системные вызовы, устанавливающие логическое размещение процессоров в виде гиперкуба или произвольного графа. Статья [44] предлагает распознавать коммуникационную структуру вычислений посредством анализа трассировочного файла. Предложенная система построена на метрике, при помощи которой из ряда шаблонов выбирается граф, наиболее близкий к коммуникационному графу программы. Возможность просмотра коммуникационного графа, вершины которого размещены на экране в соответствии с логической топологией вычислений, позволяет выявлять аномалии в передаваемых сообщениях между процессорами (неправильный адрес назначения или отправителя), а также некорректное отображение процессов на процессоры вычислительной системы.

Многие отладчики [30,25] имеют возможность строить диаграммы зависимости процессов (ось Y) от времени (ось X). Дуги на такой диаграмме характеризуют пересылку сообщений. Диаграмма Фейнмана из системы Paragraph [23] помимо коммуникационной информации предоставляет возможность оценить загрузку коммуникационных каналов.

Системы, предназначенные для вычисления производительности программы, имеют механизмы вывода статистической информации в графическом виде.

Tapestry [31] поддерживает круговые диаграммы, гистограммы, индикаторные схемы, диаграммы Кивиата и матрицы. Paragraph [23] использует диаграммы Кивиата и гистограммы для изображения статистик использования процессора и матрицы для информирования об объеме трафика сообщений.

Среда SIEVE [26], основанная на представлении производительности программы в виде электронных таблиц, может быть использована для создания XY схем. Колонки из электронной таблицы назначаются на оси X и Y. Линии и многоугольники соединяют точки на схеме. Цвет может быть использован для выделения некоторых атрибутов или идентификатора процессора.

4. Моделирование параллельной программы

В этом разделе рассматриваются вопросы моделирования параллельной программы при помощи алгебраических формул, зависящих от размерности задачи и числа процессоров. Анализируются особенности строения таких моделей и универсальность средств моделирования.

Параллельные программы создаются для вычислительных систем с большим количеством процессоров (десятки, сотни), однако тестирование и отладку обычно проводят на компьютерах, содержащих на порядки меньшее число вычислительных элементов (менее десяти). После того, как программа отлажена для эффективного выполнения на небольшом числе процессоров, разработчики встают перед дилеммой, как обеспечить эффективное выполнение на более мощной вычислительной системе и реальных размерах задачи: отлаживать программу непосредственно на мощном компьютере или моделировать выполнение программы на отладочном компьютере разработчика, используя различную трассировочную информацию.

Первый вариант является малопривлекательным, поскольку требует монопольного использования в течение продолжительного периода времени мощной вычислительной системы, время работы которой достаточно дорогое. Также для разработчика остается скрытой производительность программы на еще более мощном компьютере, т.е. такой подход не затрагивает вопросы масштабируемости программы. Поэтому разработчикам необходимы инструменты, позволяющие моделировать параллельную программу с целью анализа времени ее выполнения и масштабируемости.

Моделирование параллельной программы состоит в построении модели, дающей возможность вычислять (предсказывать) время выполнения программы T или ее масштабируемость как функцию, зависящую от размерности задачи N и числа процессоров P, т.е. необходимо найти такую функцию F, чтобы время выполнения $T=F(N,P)$ вычислялось с наименьшей погрешностью. Как правило, N является векторной величиной, содержащей, например, для задач математической физики, шаг сетки по горизонтали и вертикали. При анализе масштабируемости обычно вычисляется асимптотическое ускорение, т.е. ускорение, в котором не накладывается ограничение на доступное число процессоров, или какая-либо другая метрика

масштабируемости [7]. Учитывая тесную связь между временем выполнения и масштабируемостью параллельной программы можно считать, что такие варианты моделирования решают единую задачу улучшения масштабируемости параллельной программы и сокращения временных потерь, возникающих в ходе ее выполнения.

4.1 Средства автоматизированного моделирования

Средства моделирования программ используют один из следующих подходов:

- алгебраический анализ;
- абстрактная интерпретация;
- симулирование выполнения.

Алгебраический анализ заключается в математическом моделировании вычислительных и коммуникационных операций в программе [8]. Абстрактная интерпретация заменяет алгебраическими выражениями только вычислительные блоки кода, симулируя все коммуникационные операции [9]. Подход симулирования выполнения эмулирует (интерпретирует) каждую инструкцию программы [10]. Несмотря на то, что последние два варианта в полном объеме не строят модели программы, абстрактная интерпретация и симулирование выполнения позволяют получать детальные трассировки поведения программы без ее непосредственного выполнения на большой вычислительной системе. Это является особенно важным, когда такая система не доступна программисту для проведения полной отладки и доводки параллельной программы.

Важно заметить, что любая модель может рассматриваться только в контексте конкретной вычислительной системы. Если в самой модели не заложены механизмы независимости от конкретной архитектуры, то при переносе программы на другую компьютерную систему необходимо провести повторное моделирование.

Получение информации о производительности моделируемой программы на данной вычислительной системе осуществляется, как правило, посредством прогонки специальных тестов (бенчмарков), одноразового выполнения инструментированной версии программы или при помощи каких-либо других специализированных механизмов, например, таких как бенчмапы [11].

Моделирование SPMD программ является более простой задачей, чем общий случай параллельной программы для MIMD архитектуры. Для SPMD программ возможно применение более грубых подходов в моделировании с использованием алгебраических абстракций приложения и компьютерной системы без значительных потерь в точности [12].

Статья Сарукая [13] описывает методологию и набор средств (MK-toolkit) для автоматизированного анализа масштабируемости параллельных программ, основанных на парадигме передачи сообщений. Суть примененного метода

состоит в получении информации о программе посредством анализа исходных текстов и построения дерева синтаксического разбора, а также анализа трассировочных файлов, получаемых в ходе выполнения инструментированной версии программы и иллюстрирующих коммуникационные взаимодействия процессов.

Для каждого элемента дерева, представляющего собой цикл, условие ветвления, подпрограмму, вызов функции или коммуникационную операцию МК определяет формулу вычисляющую время выполнения блока программы и принимающую в качестве параметров размерность задачи N и количество процессоров P. Такие формулы, где это удается МК, определяются автоматически, используя трассировочные файлы от последовательных запусков программы на разных исходных данных. В противном случае, пользователь должен вручную задавать зависимость времени выполнения данного блока программы от параметров N и P. Таким образом, программа делится на набор блоков, для каждого из которых формулируется своя функция зависимости времени выполнения от параметров N и P. MK-toolkit учитывает то, что коммуникационные операции могут происходить параллельно с вычислениями посредством так называемых коммуникационных шаблонов. Авторы проверяли работоспособность модели на разных прикладных задачах, таких как решение трехдиагональных систем, симулирование атмосферных явлений и обнаружили, что погрешность моделирования составляет в среднем 15%.

Брехм и другие ставили перед собой несколько иную задачу [14], - оценить точность алгебраических моделей параллельной программы на примере PSTSWM программы [42]. В статье поставлена задача оценить при помощи моделирования, какие варианты распараллеливания наиболее эффективны для данной задачи, насколько детальными должны быть модели, чтобы точность моделирования была достаточно высокой, и как будет меняться погрешность моделирования при изменении параметров задачи и числа доступных процессоров. По аналогии с MK-toolkit, предложенная утилита PerPreT описывает исходную SPMD программу как набор формул характеризующих вычислительную (арифметические выражения) и коммуникационную (вызовы библиотеки межпроцессных взаимодействий) части программы. Особенность PerPreT заключается в том, что модель программы и вычислительной системы отделены друг от друга, тем самым при переносе программы на компьютер другой архитектуры не требуется изменять модель программы.

Авторы моделировали PSTSWM программу с разность степенью детализации модели и охарактеризовали те аспекты выполнения данной программы, которые должна учитывать модель для достижения достаточно малой (менее 5%) погрешности моделирования.

4.2 Анализ эффективности

Рассматривая средства моделирования времени выполнения и масштабируемости параллельных программ можно выделить общие аспекты для всех таких утилит:

- вся программа делится на две крупных части: вычислительная и коммуникационная, которые фактически моделируются отдельно;
- вычислительная часть разбивается на элементарные блоки (ветвления, циклы, подпрограммы и пр.), которые моделируются, как правило, отдельно;
- каждый блок описывается в модели алгебраическим выражением, зависящим от нескольких параметров. В их число входят как статические параметры, такие как размерность задачи и число процессоров, так и динамические - вероятность ветвления, число итераций в цикле, размер передаваемого сообщения;
- вид алгебраического выражения вычисляется либо в ходе тестовых прогонов инструментированной программы на разных исходных данных, либо эмпирически задается пользователем на основе досконального знания данной предметной области и моделируемой программы;
- динамические параметры (вероятность ветвления, число итераций в цикле, размер передаваемого сообщения) в алгебраических выражениях модели вычисляются в ходе тестовых прогонов инструментированной версии программы;
- алгебраические выражения для коммуникационной части чаще всего зависят от размера и режима передачи сообщений: синхронный, асинхронный, широковещательная рассылка, операции редукции, синхронизация и т.д. Режимы передачи определяются в ходе синтаксического анализа исходных текстов программы.

Коммуникационную подсистему можно достаточно точно моделировать независимо от конкретного приложения [15]. Все коммуникационные операции можно определить через время подготовки, время передачи одного байта, размер сообщения, количество процессов, участвующих в групповой операции, скорость обмена данными в памяти (для буферизованных операций) и некоторых других. Полученные формулы можно использовать в средствах моделирования без необходимости производить большое количество тестов для определения быстродействия коммуникационной операции в отлаживаемой программе.

Средства моделирования программ дают возможность оценить ее время выполнения и масштабируемость при разных размерностях задачи, без необходимости прогонов на вычислительной системе. С другой стороны они обладают рядом существенных недостатков:

- требуется серьезное вмешательство разработчика в процесс построения модели, поскольку большая часть задач, возникающих при построении модели, не может решаться автоматически;
- средства моделирования разрабатываются и тестируются для решения конкретного спектра задач (например, прогнозирование погоды), поэтому на других задачах они намного менее эффективны. Это связано с тем, что в одних задачах могут преобладать циклы с постоянными границами, в других - с переменными, или границы вложенного цикла зависят от номера итерации объемлющего цикла;
- все модели пренебрегают теми или иными аспектами выполнения параллельной программы. Например, Брехм исходит из того, что все процессы SPMD программы выполняются синхронно. Такое предположение может привести к большой погрешности моделирования при анализе многих задач. Такие предположения также влияют на стабильность погрешности моделирования. При различных входных данных (размерность задачи и число процессоров) погрешность варьируется от 0 до 27 процентов.

На сегодняшний день средства моделирования времени выполнения и масштабируемости параллельных программ не являются универсальными. Все утилиты проектировались для анализа конкретного спектра задач, как правило, моделей физических явлений. Как результат, применение данных средств к другим классам задач приводит к увеличению погрешности моделирования.

5. Отладка (мониторинг) параллельной программы

В этом разделе рассматриваются вопросы отладки (мониторинга) параллельной программы как контролируемого выполнения. На базе черновой версии стандарта анализируются отличительные особенности параллельного отладчика. Дается обзор таких отладчиков как P2D2, разрабатываемый в лабораториях NASA, Prism, входящих в пакет SUN HPC ClusterTools, и TotalView компании Etnus.

SPMD программа отлаживается в два этапа - как последовательная и как параллельная. На первом этапе используется традиционный отладчик, под контролем которого выполняется одна копия программы (SPMD программа должна корректно функционировать в однопроцессном варианте). На втором этапе программа (запущено несколько копий) отлаживается с использованием параллельного отладчика. На сегодняшний день функциональные возможности параллельного отладчика не стандартизированы, доступен только черновой вариант стандарта [6].

Отладчик - это средство, дающее пользователю возможность наблюдать и контролировать выполнение отлаживаемой программы, так называемой целевой программы. Параллельный отладчик выполняет данную функцию для параллельной программы.

5.1 Особенности архитектуры параллельного отладчика

Параллельная программа состоит из одного или более процессов, каждых из которых ассоциирован с исполняемым файлом (в SPMD модели единым для всех процессов). Каждый процесс, в свою очередь, как правило, стоит из нескольких нитей. Таким образом, целевая программа представляет собой множество нитей и/или взаимодействующих процессов.

Важным для отладчика является поддержка отладки в терминах исходного языка, т.е. отладчик должен не только контролировать выполнение целевой программы на уровне регистров и памяти, но и поддерживать высокоуровневый интерфейс в терминах исходного языка, выраженный в использовании переменных и функций целевой программы.

Пользователь взаимодействует с отладчиком посредством команд, которые для удобства могут быть обрамлены графическим интерфейсом. Несмотря на то, что отладчик может поддерживать как последовательную (следующая команда может выполняться только после того, как закончена предыдущая), так и параллельную (допускается параллельное выполнение некоторых команд) модель команд, существенным является конкретизация момента, когда команда считается оконченной. Для некоторых команд (вывести на экран значения элементов массива) такой момент очевиден. Для других команд (пошаговое выполнение) момент окончания не является интуитивно понятным. Если процесс на данном шаге выполняет функцию принятия сообщения, то это может потребовать у него перехода в режим ожидания, пока сообщение не придет от другого процесса. Можно ли считать шаг завершенным, если процесс ожидает приема сообщения? Таким образом, для каждой команды отладчика следует явно указывать условия, когда данная команда может быть выполнена (предусловие), и момент ее окончания.

Управление выполнением последовательной программы не составляет трудностей, поскольку целевая программа может находиться либо в состоянии останова, либо в состоянии выполнения. Если в отлаживаемой программе возникает событие, то программа останавливается. Пользователь может исследовать ее состояние, адресное пространство и потом продолжить выполнение. Управление параллельной программой существенно более сложное, поскольку каждая нить целевой программы имеет свое уникальное состояние выполнения. При возникновении события следует решить, что должно быть сделано с другими нитями.

Многие параллельные отладчики (P2D2 [32], TotalView [33], CXdb [34], Prism [35]) поддерживают выполнение операций с группами процессов (нитей). Группировка процессов (нитей) позволяет адресовать команду одному, многим или всеми процессам (нитям) в целевой программе. Аналогично, реакция на событие, возникшее в некотором процессе (нити) группы может распространяться на все объекты группы. Например, если в некоторой нити была достигнута точка останова (breakpoint), то будут остановлены все нити, входящие в данную группу.

Следующий вопрос, в котором проявляются особенности параллельного отладчика, заключается в том, какая модель "старт-стопа" реализована в отладчике. В последовательных отладчиках, поддерживающих отладку многонитиевых (multithreaded) целевых программ, как правило, используется модель "остановить всех" (stop-the-world). При возникновении события в одной нити, останавливаются все нити процесса.

Для клиент-серверных приложений такая модель не применима: возникновение события в клиенте не должно останавливать сервер. Поэтому для многопроцессных программ используется модель "не трогать остальных" (leave-others-alone). В общем случае параллельный отладчик должен использовать смешанную модель: "остановить всех" для нитей одного процесса и "не трогать остальных" для процессов. Возникновение события в нити приводит к останову всех нитей данного процесса, но не влияет на другие процессы целевой программы.

Модель возобновления выполнения целевой программы является зеркальным отражением модели останова. По умолчанию возобновляют выполнение только остановленные нити, попавшие в одно множество (все остановленные нити в результате возникновения события). Параллельный отладчик может также поддерживать возможность возобновления отдельных нитей (например, группы нитей) или всех нитей целевой программы.

5.2 Обзор реализаций

Параллельный отладчик может проектироваться двумя способами:

- независимо (с нуля) (TotalView [33], Prism [35], CXdb [34]);
- надстройка для последовательного отладчика (P2D2 [32], GUARD [43]).

Во втором случае для каждого процесса целевой программы запускается отдельная копия многонитиевого отладчика. В качестве последовательного отладчика, как правило, используется GDB. Основной причиной для этого является наличие реализаций GDB для большого количества вычислительных платформ, что позволяет легко переносить отладчик на другие платформы, а также создавать отладчик для гетерогенных вычислительных систем.

5.3 P2D2 (Portable Parallel / Distributed Debugger)

В статье [32] описан параллельный отладчик для гетерогенных вычислительных систем P2D2, построенный по технологии клиент-сервер. Отладочная часть сервера основывается на свободно распространяемом отладчике GDB.

Основной задачей при создании P2D2 являлась задача построения масштабируемого параллельного отладчика для гетерогенных вычислительных систем. Под масштабируемостью в данном контексте понимается масштабируемость пользовательского интерфейса, т.е. способность отладчика

предоставлять пользователю возможность исследовать состояние целевой программы (просматривать значения переменных, исходные тексты, стэк и другие компоненты вычислительного состояния программы) и контролировать ее выполнение (останавливать выполнение в заданных точках и возобновлять его после исследования состояния программы) вне зависимости от числа выполняющихся в ней процессов. Отладка гетерогенных вычислительных систем достигается за счет использования отладчика GDB, доступного для многих вычислительных платформ, а также клиент-серверного строения P2D2, позволяющего достаточно легко адаптировать отладчик для новой платформы. Масштабируемость пользовательского интерфейса отладчика обеспечивается иерархичностью информации, предоставляемой пользователю. P2D2 определяет три уровня абстракции для просмотра состояния целевой программы:

- решетка процессов (process grid) - верхний уровень;
- выделенная группа (focus group) - средний уровень;
- выделенный процесс (focus process) - нижний уровень.

Решетка процессов представляет собой область экрана, в которой представлены все процессы целевой программы в виде иконок. Пользователь имеет возможность программировать внешний вид процессов в решетке (тип иконки), в зависимости от состояния целевой программы. Такая настройка достигается посредством определения пользователем набора предикатов, проверяемых в каждом отлаживаемом процессе. В зависимости от значения предикатов процесс отображается соответствующей иконкой в решетке процессов. Например, выполняющиеся процессы имеют по умолчанию зеленую иконку, а остановленные - красную.

В области экрана, отвечающей выделенной группе, отображается более детальная информация о процессах, сгруппированных пользователем. Эта область представляет собой список, в котором каждому процессу отведена одна строка. Для каждого процесса группы отображается такая информация как идентификатор процесса, наименование машины, на которой выполняется процесс, статус процесса (выполняется, остановлен и т.п.), расшифровка статуса (адрес, по которому остановлен процесс) и т.д.

Область для просмотра состояния выделенного процесса предназначена для показа исходных текстов целевой программы.

5.4 TotalView

Параллельный отладчик TotalView [33] выделяется тем, что он поддерживается руководством ускоренной суперкомпьютерной инициативы (ASCI).

По своим функциональным возможностям TotalView во многом аналогичен P2D2, но имеет ряд особенностей:

- автоматическое подключение к отладочной среде параллельных процессов целевой программы (TotalView отслеживает создание процессов);
- автоматическое добавление новых процессов и нитей в соответствующие группы;
- позволяет отлаживать программы, не содержащие отладочной информации. В такой ситуации программа представляется пользователю на уровне машинных кодов;
- TotalView позволяет изменять исходные тексты программы (на некоторых платформах даже машинные коды) во время отладки. Это сокращает время поиска и исправления ошибок;
- TotalView имеет встроенный интерфейс командной строки (CLI) для тех случаев, когда нет возможности использовать графический интерфейс пользователя;
- поддерживает исследование core файла, полученного в результате аварийного останова программы;
- пользователь может изменять реакцию отладчика на сигналы в целевой программе;
- встроенный визуализатор данных (Visualizer) позволяет графически отображать массив данных, динамический граф вызова процедур в программе, граф передачи сообщений между нитями.

В то же время, в отличие от P2D2, TotalView не поддерживать отладку гетерогенных программ. Все узлы вычислительной системы, на которых выполняется целевая программа, должны иметь одинаковую архитектуру и операционную систему.

Основная информация в TotalView о целевой программе представляется на трех экранах:

- корневое окно (Root window);
- окно процессов (Process window);
- окно переменных (Variable window).

Корневое окно представляет общую информацию о целевой программе. Окно состоит из нескольких закладок, на которые выводится информация об отлаживаемых процессах и нитях, процессах, к которым TotalView может подключиться для отладки, группах, сформированных пользователем для управления процессами и нитями, а также журнал всех действий пользователя за время отладочной сессии.

Окно процессов является основным окном отладочной сессии. В нем отображается подробная информация о процессе и нитях внутри этого

процесса. Панели внутри окна отображают трассу стэка, фрейм стэка, исходный текст для выделенной нити, информацию о контрольных точках и т.д.

Окно переменных предоставляет информацию об адресе, типе и значении локальных, а также глобальных переменных. В этом окне также можно посмотреть значения ячеек памяти в заданном диапазоне адресов.

TotalView предоставляет широкие возможности по спецификации контрольных точек в целевой программе. Помимо стандартных возможностей, таких как точки останова по адресу, барьерные точки останова (аналогично MPI_Barrier), условных точек останова (нить останавливается, если выполнено выражение), точки останова при изменении значения переменной, TotalView позволяет вставлять во время сеанса отладки фрагменты кода по заданному адресу. Такие вставки могут содержать любые операторы, включая вызовы функций, определенных в данном процессе. Пользователь может писать их на C, C++, Фортране и ассемблере. TotalView на лету компилирует данный фрагмент кода и учитывает его во время отладки. Благодаря этой возможности пользователь может динамически исправлять ошибки в программе во время отладки, без необходимости изменять код, компилировать программу и начинать новую отладочную сессию.

6. Заключение

Анализ распределенной параллельной программы является одной из наиболее актуальных задач в области современного параллельного программирования. Природа параллельной программы требует наличия развитых средств отладки, таких как программы мониторинга, утилиты, позволяющие выполнять программу детерминировано, визуализировать ее выполнение и обмен сообщениями между процессами, оценивать ее масштабируемость при помощи соответствующих алгебраических моделей, устранять лишние временные задержки, возникающие в ходе выполнения программы.

В этой работе представлен обзор задач, возникающих при отладке, анализе и доводке распределенной масштабируемой SPMD программы, а также методов и средств решения поставленных задач.

К сожалению, на сегодняшний день не существует средств в полной мере решающих все стоящие задачи. Однако повышенный интерес коммерческих компаний и исследовательских центров, а также большое число академических проектов и развитие отраслевых и индустриальных стандартов свидетельствуют о том, что в ближайшем будущем данная проблема будет решена.

Литература

1. Xian-Xe Sun, The Relation of Scalability and Execution Time, Proc. of the 10th Int Parallel Processing Symposium, 1996.

2. Amdahl, G.M., Validity of the single-processor approach to achieving large scale computing capabilities. In AFIPS Conference Proc. vol. 30 (Atlantic City, N.J., Apr. 18-20). AFIPS Press, Reston, Va., 1967, pp. 483-485.
3. Gustafson, J.L., Reevaluating Amdahl's Law, CACM, Vol. 31, No. 5, 1988. pp. 532-533.
4. Damodaran-Kamal S.K., Francioni J.M., Nondeterminacy: Testing and Debugging in Message Passing Parallel Programs, Proc. of the ACM/ONR Workshop on Parallel and Distributed Debugging, ACM SIGPLAN Notices, Vol.28, No.12, Dec. 1993, pp.118-128.
5. Sarukkai S.R., Malony A., Perturbation analysis of high level instrumentation for SPMD programs, 4th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, 1993, pp. 44-53.
6. www.ptools.org/hpdf/draft.
7. Sahni, S., Thanvantri, V., Performance Metrics: Keeping the Focus on Runtime, IEEE Parallel and Distributed Technology, Vol. 4, No. 1, 1996, pp., 43-56.
8. Sarukkai, S.R., Scalability Analysis Tools for SPMD Message-Passing Parallel Programs, Proc. of the Second Int Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Mascots'94), CS Press, 1994, pp. 180-186.
9. Mehra, P., Gower, M., Bass, M., Automated Modeling of Message-Passing Programs, Proc. of the Second Int Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Mascots'94) , CS Press, 1994, pp. 187-192.
10. Mehra, P., Schulbach, C., Yan, J., A Comparison of Two Model-Based Performance-Prediction Techniques for Message-Passing Parallel Programs, Sigmetrics Conference on Measurement and Modeling of Computer Systems, ACM Press, New York, 1994, pp. 181-190.
11. Toledo, S., Performance Prediction with Benchmaps, Proc. ot the 10th Int Parallel Processing Symposium, 1996.
12. Brehm J., Dowdy L., Madhukar M., Smirni E., PerPreT - a performance prediction tool, in Quantitative Evaluation of Computing and Communication Systems, Lecture Notes in Computer Science 977, Springer, Heidelberg, 1995.
13. Sarukkai S.R., Mehra P., Block R. J., Automated scalability analysis of message-passing parallel programs, IEEE Parallel and Distributed Technology, 3 (1995), pp. 21-32.
14. Brehm J., Worley P.H., Performance Prediction for Complex Parallel Applications, Proc. of the 11th Parallel Processing Symposium, 1997.
15. Rauber T., Runger G., Modeling the Runtime of Scientific Programs on Parallel Computers, Proc. ot the 2000 Int Workshops on Parallel Processing, 2000.
16. Miller B.P., What to Draw? When to Draw? An Essay on Parallel Program Visualization. Journal of Parallel and Distributed Computing, Vol. 18, No. 2, June 1993, pp. 265-269.
17. Geist G.A., Heath M.T., Peyton B.W., Worley P.H., PICL: A portable instrumented communication library, C reference manual, ORNL Technical Report ORNL/TM-11130, July 1990.
18. Malony A.D., Reed D.A., Arendt J.W., Aydt R.A., Grabas D., Totty B.K. An integral performance data collection, analysis and visualization system. Proc. of the Fourth Conference on Hypercube Concurrent Computers and Applications. Monterey, CA, Mar. 1989.
19. Kimelman D., Ngo T., Program Visualization for RP3: An overview. Technical report, IBM Research, Division. T.J.Watson Research Center, 1990.
20. Sarukkai S.R., Performance visualization and prediction of parallel supercomputer programs: An interim report. Technical report 318. Indiana University, Bloomington, IN, Nov. 1990.
21. Reed D.A., Aydt R.A., Madhyastha T.M., Noe R.J., Shields K.A., Schwartz B.W. The Pablo Performance Analysis Environment. Technical report, University of Illinois at Urbana, Champaign, Department of Computer Science, Nov. 1992.
22. Miller B.P., Callaghan M.D., Cargille J.M., Hollingsworth J.K., Irvin B.R., Karavanic K.L., Kunchithapadam K., Newhall T. The Paradyn Performance Measurement Tool, Computer, Vol. 28, No. 11, Nov. 1995, pp. 37-46.
23. Health M.T., Etheridge J.A., Visualizing the Performance of Parallel Programs, IEEE Software Vol. 8, No. 5, Sept. 1991, pp. 29-39.
24. Lamport, L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM., Vol. 21, No. 7, July 1978, pp. 558-565.

25. LeBlanc T.J., Mellor-Crummey J.M., Fowler R.J. Analyzing parallel program execution using multiple views. Journal of Parallel and Distributed Computing, Vol. 9, No. 2, June 1990, pp. 203-217.
26. Sarukkai S.R., Gannon D. Performance visualization of Parallel Programs using SIEVE.1. Proc. of the 1992 Int Conference on Supercomputing. Washington. D.C., July 1992, pp. 157-166.
27. Stone J.M., A graphical representation of concurrent processes, SIGPLAN Notices, Vol. 24, No. 1, Jan. 1989, pp. 226-235 [Proc. of the Workshop on Parallel and Distributed Debugging, Madison, WI, May 1988].
28. Zernick D., Rudolf L., Animating work and time for debugging parallel programs - Foundations and Experience. SIGPLAN Notices, Vol. 26, No. 12, Dec. 1991, pp. 46-56 [Proc. of the SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, Santa Cruz, CA, May 1991].
29. Beguelin A., Dongarra J.J, Geist G., Manchek R., Sunderam V., Graphical development tools for network-based concurrent supercomoputing. Proc. of Supercomputing'91. Albuquerque, New Mexico, Nov. 1991, pp. 431-444.
30. Harter P.K., Heimbigner D.M., King R. IDD: An interactive distributed debugger, Proc. of the Fifth Int Conference on Distributed Computing Systems, May 1985, pp.498-506.
31. Malony A.D., Reed D.A. Visualizing parallel computer system performance. In Simmons M., Koskela R., Bucher I., (Eds.) Parallel Computer Systems: Performance Instrumentation and Visualization. Association for Computer Machinery, New York, 1990.
32. Hood R., The p2d2 Project: Building a Portable Distributed Debugger, Proc. of the SIGMETRICS Symposium on Parallel and Distributed Tools, May 1996.
33. Etnus, Inc. The TotalView Multiprocess Debugger, www.etnus.com/products/totalview
34. Convex Computer Corporation, Convex CXdb User's Guide, Second Edition, October 1993, DSW-473.
35. docs.sun.com/ab2/coll.514.2/PRISMUG.
36. Heath M.T., Malony A.D., Rover D.T., Parallel Performance Visualization: From Practice to Theory, IEEE Parallel and Distributed Technology, Vol.3, No.4, 1995, pp. 44-60.
37. Bakic A.M., Mutka M.W., Rover D.T., BRISK: A Portable and Flexible Distributed Instrumentation System, Michigan State University, Technical Report.
38. Zheng Q., Chen G., Huang L., Optimal Record and Replay for Debugging of Nondeterministic MPI/PVM Programs, Proc. of the Fourth Int Conference /Exhibition on High Performance Computing in Asia-Pacific Region, 2000.
39. Paik E.H., Chung Y.S., Lee B.S., Yoo C.-W., A Concurrent Program Debugging Environment using Real-Time Replay, Proc. of the 1997 Int Conference on Parallel and Distributed Systems.
40. Zhang K., Sun C., Li K.-C., Dynamically Instrumenting Message-Passing Programs Using Virtual Clocks, Macquarie University, Technical Report.
41. Taylor V.E., Wu X., Geisler J., Li X., Lan Z., Stevens R., Hereld M., Judson I.R., Prophesy: An Infrastructure for Analyzing and Modeling the Performance of Parallel and Distributed Applications, Proc. of the 9th IEEE Int Symposium on High Performance Distributed Computing.
42. Worley P.H.,Toonen B., A users' guide to PSTSWM, Tech. Report ORNL/TM-- 12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.
43. Abramson D.A., Sosic R., A Debugging and Testing Tool for Supporting Software Evolution, Journal of Automated Software Engineering, 1996, Vol.3, pp. 369 – 390.
44. Huband S., McDonald C., Debugging parallel programs using incomplete information, Proc. of the 1st IEEE Computer Society Workshop on Cluster Computing, pp. 278 - 286, IEEE Computer Society Press, 1999.